ODYSSEUS/EduCOSMOS Project #2: EduOM Project Manual

Version 1.0

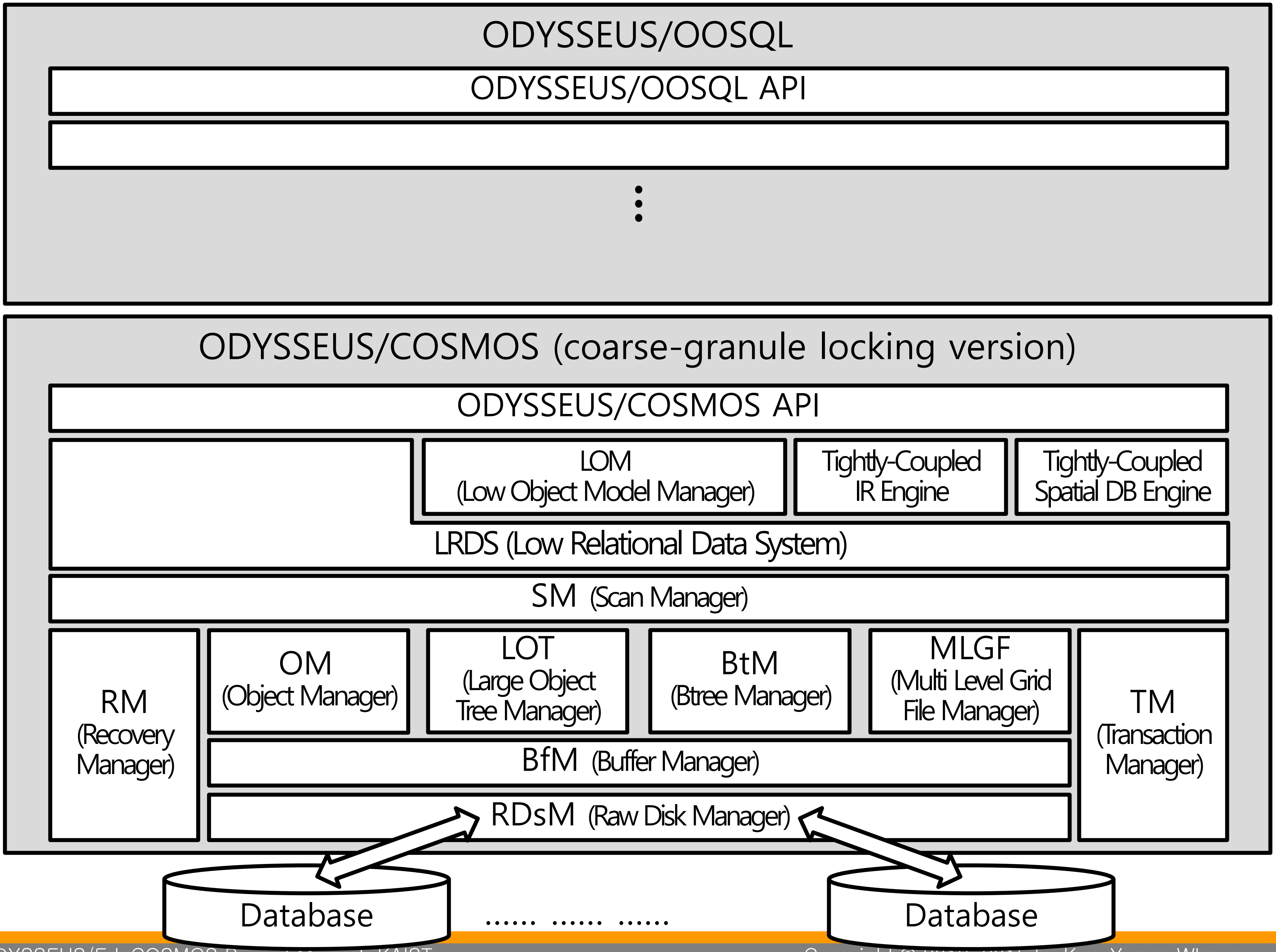
Contents

ODYSSEUS/COSMOS

- ODYSSEUS
  - An object-relational DBMS developed by Kyu-Young Whang et al. at Advanced Information Technology Research Center (AITrc) / Computer Science Department of KAIST. ODYSSEUS has been being developed since 1990.

- ODYSSEUS/COSMOS
  - The storage system of ODYSSEUS, which is used as an infrastructure for various database application softwares.

- ODYSSEUS architecture


ODYSSEUS/OOSQL
ODYSSEUS/OOSQL API
⋮
ODYSSEUS/COSMOS (coarse-granule locking version)
ODYSSEUS/COSMOS API
LOM (Low Object Model Manager)
Tightly-Coupled IR Engine
Tightly-Coupled Spatial DB Engine
LRDS (Low Relational Data System)
SM (Scan Manager)
RM (Recovery Manager)
OM (Object Manager)
LOT (Large Object Tree Manager)
BtM (Btree Manager)
MLGF (Multi Level Grid File Manager)
TM (Transaction Manager)
BfM (Buffer Manager)
RDsM (Raw Disk Manager)
Database
....
Database

ODYSSEUS/EduCOSMOS Project

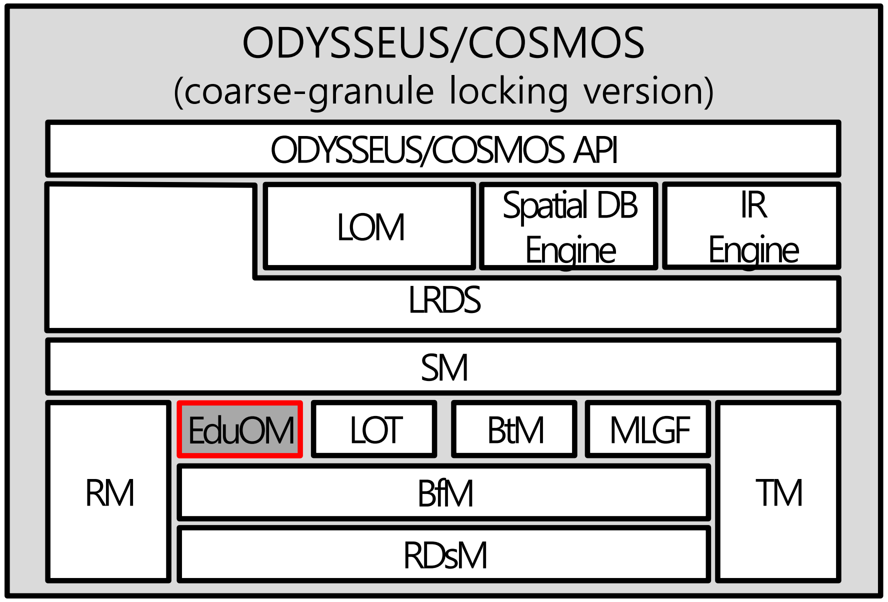
- Overview
  - A project for educational purposes where students implement a part of the coarse-granule locking version of the ODYSSEUS/COSMOS storage system
    - Prerequisites for the project: basic C programming skill

- Objective
  - To learn the functions of each module of a DBMS by implementing a part of the ODYSSEUS/COSMOS storage system

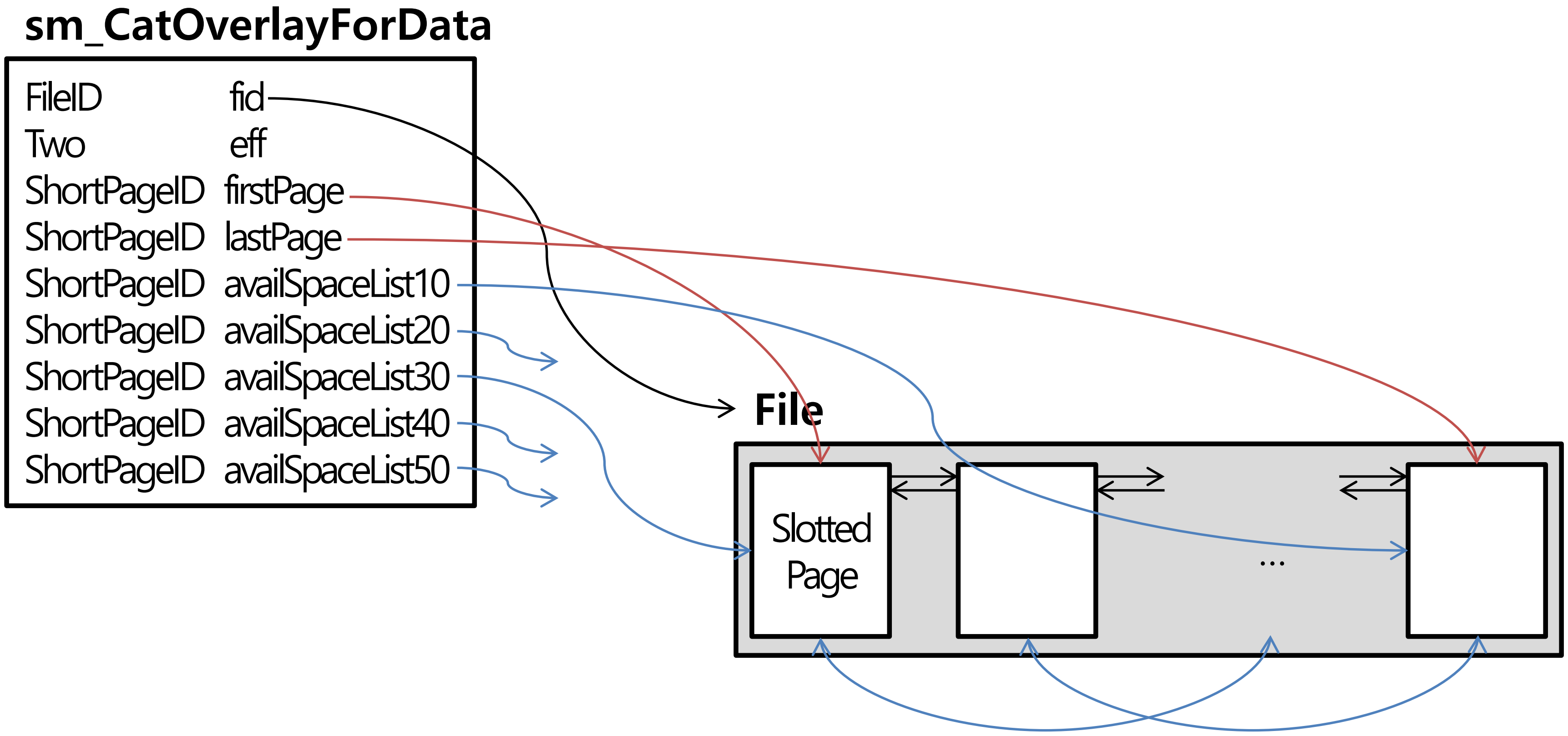
- Project types
  - EduBfM
    - We implement the operations of the buffer manager.
  - EduOM
    - We implement the operations of the object manager and the page-related structures.
  - EduBtM
    - We implement the operations of the B+ tree index manager.

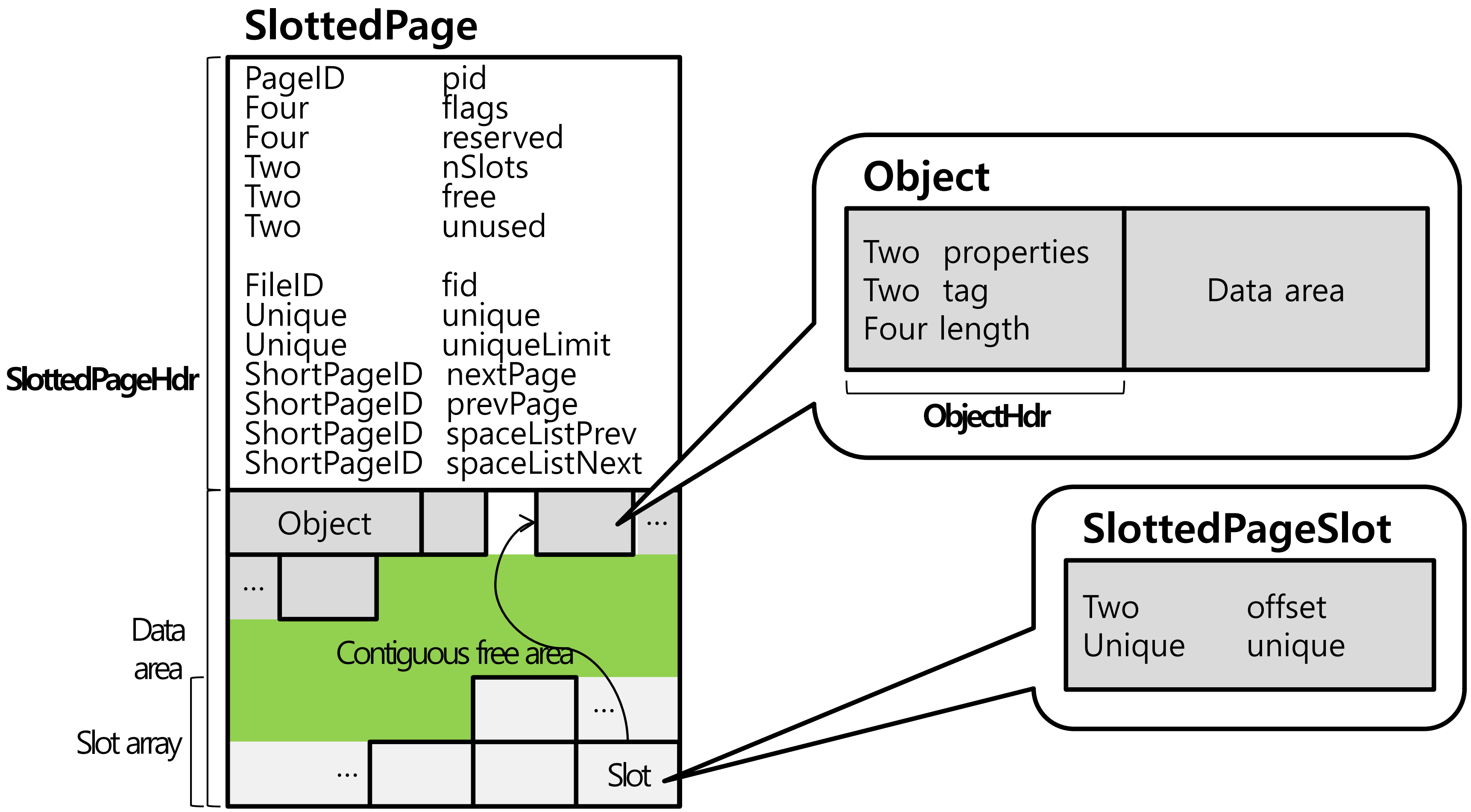
EduOM Project

- Objective
  - We implement the operations of the slotted page related structure to store objects.
  - In EduOM, we handle only a very limited subset of original ODYSSEUS/COSMOS OM functionality.



Data Structures


sm_CatOverlayForData
FileID fid
Two eff
ShortPageID firstPage
ShortPageID lastPage
ShortPageID availSpaceList10
ShortPageID availSpaceList20
ShortPageID availSpaceList30
ShortPageID availSpaceList40
ShortPageID availSpaceList50
File
Slotted Page
…


SlottedPage
SlottedPageHdr
PageID pid
Four flags
Four reserved
Two nSlots
Two free
Two unused
FileID fid
Unique unique
Unique uniqueLimit
ShortPageID nextPage
ShortPageID prevPage
ShortPageID spaceListPrev
ShortPageID spaceListNext
Object
…
Contiguous free area
Data area
Slot array
Slot
Object
Two properties
Two tag
Four length
Data area
ObjectHdr
SlottedPageSlot
Two offset
Unique unique

sm_CatOverlayForData

- Overview
  - A data structure to store information about the data file
    - Data file: A set of pages storing related objects
  - This information is kept separately for each data file in ODYSSEUS/COSMOS.

- Components
  - fid
    - ID of the file
  - eff
    - Extent fill factor of the file, which is not used in EduOM (You may ignore it when implementing the EduOM function.)
  - firstPage
    - Page number of the first page of the file
  - lastPage
    - Page number of the last page of the file

- availSpaceList10 ~ availSpaceList50
  - Page number of the first (most recently inserted) page of each available space list
    - 10% available space list
      - A doubly linked list of pages whose size of the remaining free space is 10~20% of the page
    - 20% available space list
      - A doubly linked list of pages whose size of the remaining free space is 20~30% of the page
    - 30% available space list
      - A doubly linked list of pages whose size of the remaining free space is 30~40% of the page
    - 40% available space list
      - A doubly linked list of pages whose size of the remaining free space is 40~50% of the page
    - 50% available space list
      - A doubly linked list of pages whose size of the remaining free space is 50% or more of the page

SlottedPage

- Overview
  - The page data structure to efficiently store and manage objects

- Components
  - header
    - The page header storing information about the page
  - data[]
    - The data area storing objects
  - slot[1]
    - The array of slots storing information required for identifying objects stored in the page.
      - Other slots (*slot*[-1] ~ *slot*[-*n*]) except the first slot (*slot*[0]) share the data area and the memory space.
        - » Array index of the first slot = 0
        - » Array index of the next slot = Array index of the previous slot – 1
        - » Slot number = |Array index of the slot|

SlottedPageHdr

- Overview
  - The data structure to store information about a page.

- Components
  - pid
    - The ID of a page
      - Consists of the page number and the volume number.
  - flags
    - A set of bits indicating the type of a page
      - If the second bit is set (SLOTTED_PAGE_TYPE), then the page is a slotted page.
      - Other bits are not used in EduOM. (You may ignore them when implementing the EduOM function.)
  - reserved
    - A reserved variable to store additional information about a page

- nSlots
  - The size of the page's slot array
    (= The last slot number among the slots currently being used + 1)
    - To efficiently use the space in a page, the size of the slot array is dynamically changed as an object is inserted/deleted.
- free
  - The starting offset of the contiguous free area in the page's data area
    - Contiguous free area: Continuous free space after the last object in the data area
- unused
  - The sum of the sizes of free spaces except the contiguous free area in the page's data area (unit: # of bytes)
- fid
  - The ID of the file including the page

- unique
  - The recently assigned unique number in a page
    - Unique number: A unique number assigned to each object
    - Initialize it to 0 at first, and call om_GetUnique() to update the value
- uniqueLimit
  - The maximum value of a unique number currently assignable in a page
    - Initialize it to 0 at first, and call om_GetUnique() to update the value.
- nextPage / prevPage
  - The page number of the next/previous page in the same file
    - Used for maintaining the doubly linked list structure between pages of a file.
- spaceListPrev / spaceListNext
  - The page number of the next/previous page in the same available space list
    - Used for maintaining the doubly linked list structure between pages of the available space list.

SlottedPageSlot

- Overview
  - The data structure to store information required for identifying the objects stored in a page

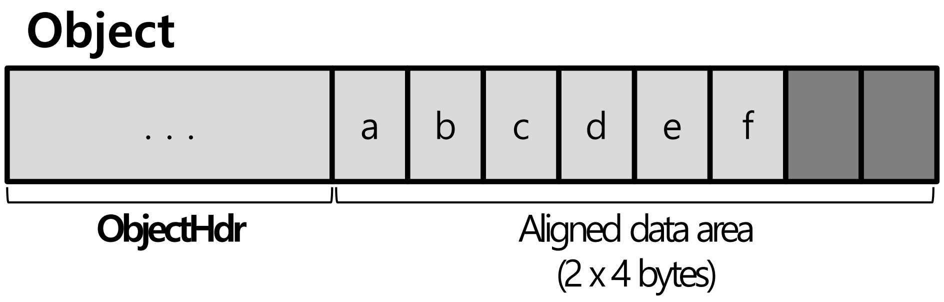
- Components
  - offset
    - The offset in the data area of an object stored in the page
      - In case of an unused slot, *offset* := EMPTYSLOT
  - unique
    - The unique number of the object
      - Call om_GetUnique() to get a unique number.

Object

- Overview
  - The data structure representing an object stored in a page
  - In EduOM, we handle only the small object whose size (size of header + size of aligned data area) is smaller than that of the data area of a page.

- Components
  - header
    - The object header storing information about the object

- data[]
  - Data area storing the object’s data
  - The size of the data area storing the data is aligned to be a multiple of 4 (the basic unit of memory allocation in a 32-bit operating system).
    - Example) Allocating the data area where the data of size 6 is stored,



ObjectHdr

- Overview
  - The data structure to store information about an object

- Components
  - properties
    - A set of bits indicating properties of an object
      - If every bit is 0, it indicates a small object.
      - Other properties are not used in EduOM. (You may ignore them when implementing the EduOM function.)
  - tag
    - A tag of an object; not used in EduOM (You may ignore it when implementing the EduOM function.)
  - length
    - The length of data in an object (unit: # of bytes)
      - The actual length of the data stored in the data area, rather than the length of the aligned data area

Operations

- Insert a new object into the page.
- Delete an object from the page.
- Compact the data area of the page.
  - Adjust the offsets of the objects for every piece of free space in the data area to form an uninterrupted contiguous free area.
- Search the page.
  - Search for a desired object in the page.

API Functions to Implement

- EduOM_CreateObject()
- EduOM_DestroyObject()
- EduOM_CompactPage()
- EduOM_ReadObject()
- EduOM_NextObject()
- EduOM_PrevObject()

(※ API functions mean they are part of the ODYSSEUS/COSMOS API shown in p.4)
(※ API: Application Programming Interface)

EduOM_CreateObject()

- File: EduOM_CreateObject.c

- Description
  - Insert a new object into the page identical (or adjacent) to the page where the object given as a parameter resides, and return the ID of the object inserted.
    - Initialize the header of the object to be inserted.
      - *properties* := 0x0
      - *length* := 0
      - If the *objHdr* given as a parameter is not NULL,
        - » *tag* := the tag value stored in the *objHdr*
      - If the *objHdr* given as a parameter is NULL,
        - » *tag* := 0
    - Call eduom_CreateObject() to insert an object into the page, and return the ID of the object inserted.

- Parameters
  - ObjectID *catObjForFile

    (IN) ID of an object storing information (*sm_CatOverlayForData*) about the file to insert the object into
  - ObjectID *nearObj

    (IN) ID of an object, near which the new object is to be inserted
  - ObjectHdr *objHdr

    (IN) Object header storing the tag value of the object to be inserted
  - Four length

    (IN) Data length of the object to be inserted (unit: # of bytes)

- char *data
  (IN) Data of the object to be inserted
- ObjectID *oid
  (OUT) ID of the object inserted

- Return value
  - Four error code

- Related function
  eduom_CreateObject()

EduOM_DestroyObject()

- File: EduOM_DestroyObject.c

- Description
  - Delete an object from a page of the file.
    - Delete the page storing the object to be deleted from the currently available space list.
    - Set the slot corresponding to the object deleted as an empty unused slot.
      - *offset* of the slot := EMPTYSLOT
    - Update the page header.
      - If the slot corresponding to the object deleted is the last slot of the slot array, update the size of the slot array.
        - » *nSlots* := the last slot number in the slots currently being used + 1
      - Update the variables *free* or *unused* depending on the offset in the data area of the object deleted.

      - » The size of the free space obtained by deleting the object
        = sizeof(*ObjectHdr*) + Size of the aligned data area
        + Size of the free space obtained by updating the slot array size

- If the deleted object is the only object in the page, and the page is not the first page of the file,
  - – Delete the page from the list of pages of the file.
  - – Deallocate the page.
    - » Allocate a new dealloc list element from the *dlPool* given as a parameter.
      - • Dealloc list: Linked list of pages to be deallocated
    - » Store the information about the pages to be deallocated into the element allocated.
    - » Insert the element into the dealloc list as the first element.
- If the deleted object is not the only object of a page, or if the page is the first page of the file,
  - – Insert the page into the appropriate available space list.
  - ❖ The ID of the file, which is immutable, is the same as the ID of its first page. Therefore, we must not deallocate the first page even if it becomes empty.

- Parameters
  - ObjectID *catObjForFile

    (IN) ID of the object storing information (*sm_CatOverlayForData*) about the file that contains the object to be deleted
  - ObjectID *oid

    (IN) ID of the object to be deleted
  - Pool *dlPool

    (INOUT) Pool to allocate a new dealloc list element from
  - DeallocListElem *dlHead

    (INOUT) Header pointing to the first element of the dealloc list

- Return value
  - Four error code

- Related functions

  EduOM_DestroyObject(), om_FileMapDeletePage(), om_PutInAvailSpaceList(), om_RemoveFromAvailSpaceList(), BfM_GetTrain(), BfM_FreeTrain(), BfM_SetDirty(), Util_getElementFromPool()

EduOM_CompactPage()

- File: EduOM_CompactPage.c
- Description
  - Adjust the offsets of the objects for every piece of free space in the data area of the page to form an uninterrupted contiguous free area.
    - If *slotNo* given as a parameter is not NIL (-1),
      - Store all the objects in the page except objects corresponding to *slotNo* contiguously from the very front of the data area.
        - » Order of storing objects: Order of the corresponding slot numbers
      - Store the object corresponding to *slotNo* as the last object in the data area.
    - If *slotNo* given as a parameter is NIL (-1),
      - Store all the objects in a page contiguously from the very front of the data area.
        - » Order of storing objects: Order of the corresponding slot numbers
    - Update the page header.

- Parameters
  - SlottedPage *apage

    (IN) Page to be compacted
  - Two slotNo

    (IN) Slot number corresponding to the object to be stored as the last one in the data area of the page to be compacted

- Return value
  - Four error code

- No related functions

EduOM_ReadObject()

- File: EduOM_ReadObject.c
- Description
  - Read the whole or a part of the data of an object, and return a pointer to the data read.
    - Use *oid* given as a parameter to access an object.
    - Use *start* and *length* given as parameters to read data of the accessed object.
      - Read as much data as *length* from *start* in the data area of the object.
      - If *length* == REMAINDER, read data to the end.
    - Return the pointer to the corresponding data.

- Parameters
  - ObjectID *oid

    (IN) ID of an object to be read
  - Four start

    (IN) Offset to start reading in the data area of the object to be read
  - Four length

    (IN) Length of the data to be read
  - char *buf

    (OUT) Pointer to the data read

- Return value
  - Four the number of bytes read,
    or error code

- Related functions

  EduOM_ReadObject(), BfM_GetTrain(), BfM_FreeTrain()

EduOM_NextObject()

- File: EduOM_NextObject.c
- Description
  - Return ID of the object next to the current object.
    - If *curOID* given as a parameter is NULL,
      - Return the ID of the first object (one of the following objects) in the file.
        - » First object in the slot array of the first page
        - » (If the first page is empty) first object of the next page
      - If the file is empty, return EOS (End Of Scan).
    - If *curOID* given as a parameter is not NULL,
      - Search for an object corresponding to *curOID*.
      - Return ID of the object next to the object found in the slot array.
        - » If the object found is the last object of the page,
          - Return ID of the next page's first object.
        - » If the object found is the last object of the file's last page,
          - Return EOS.

- Parameters
  - ObjectID *catObjForFile
    (IN) ID of the object storing the information (*sm_CatOverlayForData*) about the file that contains the current object
  - ObjectID *curOID
    (IN) ID of the current object
  - ObjectID *nextOID
    (OUT) ID of the object next to the current object
  - ObjectHdr *objHdr
    (OUT) Header of the object next to the current object

- Return value
  - Four EOS,
    or error code

- Related functions
  BfM_GetTrain(), BfM_FreeTrain()

EduOM_PrevObject()

- File: EduOM_PrevObject.c

- Description
  - Return ID of the object just before the current object.
    - If *curOID* given as a parameter is NULL,
      - Return ID of the last object in the slot array of the file's last page.
      - If the file is empty, return EOS (End Of Scan).
    - If *curOID* given as the parameter is not NULL,
      - Search for an object corresponding to *curOID*.
      - Return ID of the object just before the object found in the slot array.
        - » If the object found is the page's first object,
          - Return the ID of the previous page's last object.
          - If the previous page is empty, return EOS.
        - » If the object found is the first object of the file's first page,
          - Return EOS.

- Parameters
  - ObjectID *catObjForFile
    (IN) ID of the object storing the information (*sm_CatOverlayForData*) about the file that contains the current object
  - ObjectID *curOID
    (IN) ID of the current object
  - ObjectID *prevOID
    (OUT) ID of the object just before the current object
  - ObjectHdr *objHdr
    (OUT) Header of the object just before the current object

- Return value
  - Four EOS,
    or error code

- Related functions
  BfM_GetTrain(), BfM_FreeTrain()

Internal Function to Implement

- eduom_CreateObject()

eduom_CreateObject()

- File: eduom_CreateObject.c

- Description
  - Insert a new object into the page identical (or adjacent) to the page where the object given as a parameter resides, and return the ID of the object inserted.
    - Calculate the size of the free space required for inserting the object.
      - sizeof(*ObjectHdr*) + Size of the aligned object data area + sizeof(*SlottedPageSlot*)
    - Select a page to insert the object.
      - If *nearObj* given as a parameter is not NULL,
        - » If there is available space in the page storing *nearObj*,
          - Select the page as the page to insert the object.
          - Delete the selected page from the current available space list.
          - Compact the selected page if necessary.

      - If there is no available space in the page storing *nearObj*,
        - Allocate a new page to insert the object into.
        - Initialize the selected page's header.
        - Insert the page as the next page of the page storing *nearObj* into the list of pages of the file.
  - If *nearObj* given as a parameter is NULL,
      - If the size of the object is less than or equal to the total size of the data area of a page, and there is any page in the available space list that has the smallest available space among the lists whose available space is suitable for the size of the free space required for inserting the object,
        - Select the first page of the corresponding available space list as the page to insert the object into.
        - Delete the page from the current available space list.
        - Compact the selected page if necessary.
      - If the above condition is not satisfied, and there is available space in the file's last page (the last page in the linked list of the pages of the file),
        - Select the file's last page as the page to insert the object into.
        - Compact the selected page if necessary.

- » Else,
  - Allocate a new page to insert the object into.
  - Initialize the page’s header.
  - Insert the page as the last page into the list of pages of the file.

- Insert an object into the selected page.
  - Update the object’s header.
    - » *length* := Length of the data
  - Copy the object into the selected page’s contiguous free area.
  - Allocate an empty or new slot of the slot array to store the information for identifying the object copied.
  - Update the page’s header.
    - » The size of free space used for inserting an object
      = sizeof(*ObjectHdr*) + Size of the aligned object data area
      + sizeof(*SlottedPageSlot*) in case of allocating a new slot
  - Insert the page into an appropriate available space list.
- Return the ID of the object inserted.

- Parameters
  - ObjectID *catObjForFile

    (IN) ID of the object storing information (*sm_CatOverlayForData*) about the file to insert the object into
  - ObjectID *nearObj

    (IN) ID of the object, near which the object is to be inserted
  - ObjectHdr *objHdr

    (IN) Object header storing the tag value of the object to be inserted
  - Four length

    (IN) Data length of an object to be inserted (unit: # of bytes)
  - char *data

    (IN) Data of an object to be inserted
  - ObjectID *oid

    (OUT) ID of the object inserted

- Return value
  - Four error code

- Related functions

  EduOM_CompactPage(), om_GetUnique(), om_FileMapAddPage(), om_PutInAvailSpaceList(), om_RemoveFromAvailSpaceList(), RDsM_PageIdToExtNo(), RDsM_AllocTrains(), BfM_GetTrain(), BfM_GetNewTrain(), BfM_FreeTrain(), BfM_SetDirty()

Given API Functions

- RDsM_PageIdToExtNo()
  - Return the extent number to which the page belongs.
    - Extent: a list of physically adjacent pages
  - Parameters
    - PageID   *pageId

      (IN) ID of the page
    - Four   *extNo

      (OUT) The extent number to which the page belongs
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(...)
{
    Four firstExt;     /* number of the first extent of the file */
    sm_CatOverlayForData *catEntry;     /* pointer to the data file catalog information */
    PhysicalFileID pFid;     /* ID of the first page in the file */
    ...
    MAKE_PHYSICALFILEID(pFid, catEntry->fid.volNo, catEntry->firstPage);

    /* Get the first extent number of the file */
    e = RDsM_PageIdToExtNo((PageID *)&pFid, &firstExt);
    if( e < 0 ) ERR( e );
    ...
}
```

- RDsM_AllocTrains()
  - Allocate a new page (*sizeOfTrain*=1) or train (*sizeOfTrain*>1) in the disk, and return the ID of the page or the first page of the train allocated.
    - Train: the structure to store the large object whose size is larger than that of the data area of a page; not used in EduOM
  - Parameters
    - Four volNo

      (IN) The volume number to which the disk in which to allocate the page/train belongs
    - Four firstExt

      (IN) The first extent number to which the first page of the file in which to allocate the page/train belongs
    - PageID *nearPID

      (IN) ID of the page to which the page/train to be allocated is to be physically adjacent on the disk
    - Two eff

      (IN) Extent fill factor of the file in which to allocate the page/train

- Four numOfTrains

  (IN) The number of pages/trains to be allocated
- Two sizeOfTrain

  (IN) Size of the train to be allocated (unit: # of pages)

  (※ set *sizeOfTrain* to 1 to allocate the page.)
- PageID *trainIDs

  (OUT) ID of the page or the first page of the train allocated

– Return value

- Four error code

– Example

```
Four eduom_CreateObject(…)
{
    PageID pid;    /* ID of a page */
    PageID nearPid;    /* ID of the page to which the page to be allocated is to be
                                physically adjacent on the disk */
    Four firstExt;    /* number of the first extent of the file */
    sm_CatOverlayForData *catEntry;    /* pointer to the data file catalog information */
    …
    /* Allocate a new page */
    e = RDsM_AllocTrains(catEntry->fid.volNo, firstExt, &nearPid, catEntry->eff, 1,
PAGESIZE2, &pid);
    if( e < 0 ) ERR( e );
    …
}
```

- BfM_GetTrain()
  - Fix a page (*sizeOfTrain*=1) or train (*sizeOfTrain*>1) to the buffer, and return the pointer to the page/train fixed to the buffer.
    - Every transaction should fix the page/train to the buffer before accessing it.
    - To fix the page/train that is newly allocated on the disk to the buffer, call BfM_GetNewTrain() rather than BfM_GetTrain() for performance reasons.
      - BfM_GetNewTrain() fixes the empty page/train to a buffer without accessing the disk since it is not necessary to access the disk to read the contents of that page/train, which is empty.
  - Parameters
    - TrainID *trainId

      (IN) ID of the page or the first page of the train to be fixed
    - char **retBuf

      (OUT) Pointer to the page/train fixed to the buffer
    - Four type

      (IN) Type of the buffer to fix the page/train to
      (※ set *type* to PAGE_BUF to fix the page.)
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(
    ObjectID    *catObjForFile, /* IN: ID of the object that contains the catalog 
information */
    …)
{
    SlottedPage *catPage;   /* pointer to a buffer */
    …
    /* Fix the page that contains the catalog object to the buffer */
    e = BfM_GetTrain((TrainID*)catObjForFile, (char**)&catPage, PAGE_BUF);
    if( e < 0 ) ERR( e );
    …
}
```

- BfM_GetNewTrain()
  - Fix a page (*sizeOfTrain*=1) or train (*sizeOfTrain*>1) that has been newly allocated on the disk to a buffer, and return the pointer to the page/train fixed to the buffer.
  - Parameters
    - TrainID   *trainId

      (IN) ID of the page or the first page of the train to be fixed
    - char   **retBuf

      (OUT) Pointer to the page/train fixed to the buffer
    - Four   type

      (IN) Type of the buffer to fix the page/train to
      (※ set *type* to PAGE_BUF to fix the page.)
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(…)
{
    PageID pid;    /* ID of a page */
    SlottedPage *apage;     /* pointer to a buffer */
    …
    /* Fix the page that has been newly allocated on the disk to the buffer */
    e = BfM_GetNewTrain(&pid, (char **)&apage, PAGE_BUF);
    if( e < 0 ) ERR( e );
    …
}
```

- BfM_FreeTrain()
  - Unfix a page (*sizeOfTrain*=1) or train (*sizeOfTrain*>1) from the buffer.
    - Every transaction should unfix the page/train from the buffer after completing the access to it.
  - Parameters
    - TrainID   *trainId

      (IN) ID of the page or the first page of the train to be unfixed
    - Four   type

      (IN) Type of the buffer the page/train is fixed to
      (※ set *type* to PAGE_BUF to unfix the page.)
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(
    ObjectID    *catObjForFile, /* IN: ID of the object that contains the catalog
information */
    ...)
{
    ...
    /* Unfix the page that contains the catalog object from the buffer */
    e = BfM_FreeTrain((TrainID*)catObjForFile, PAGE_BUF);
    if( e < 0 ) ERR( e );
    ...
}
```

- BfM_SetDirty()
  - Set the DIRTY bit indicating that a page (*sizeOfTrain*=1) or train (*sizeOfTrain*>1) stored in the buffer has been modified.
  - Parameters
    - TrainID   *trainId

      (IN) ID of the page or the first page of the train to set the DIRTY bit for
    - Four   type

      (IN) Type of the buffer the page/train is fixed to
      (※ set *type* to PAGE_BUF to set the DIRTY bit of the page.)
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(…)
{
    PageID pid;     /* ID of a page */
    …
    /* Set the DIRTY bit */
    e = BfM_SetDirty(&pid, PAGE_BUF);
    if (e < 0) ERRB1(e, &pid, PAGE_BUF);
    …
}
```

- Util_getElementFromPool()
  - Allocate memory space for a new dealloc list element from the pool, and return the pointer to the memory space.
  - Parameters
    - Pool *aPool

      (IN) Element pool to allocate the element from
    - void *elem

      (OUT) dealloc list element allocated
  - Return value
    - Four error code

– Example

```
Four EduOM_DestroyObject(…
    Pool      *dlPool,        /* INOUT: pool of the elements of the dealloc list */
    DeallocListElem *dlHead)     /* INOUT: head of the dealloc list */
{
    PageID pid;     /* ID of a page */
    DeallocListElem *dlElem;     /* pointer to the element of the dealloc list */
    …
    /* Insert the deallocated page into the dealloc list */
    e = Util_getElementFromPool(dlPool, &dlElem);
    if( e < 0 ) ERR( e );

    dlElem->type = DL_PAGE;
    dlElem->elem.pid = pid;     /* ID of the deallocated page */
    dlElem->next = dlHead->next;
    dlHead->next = dlElem;
    …
}
```

Given Functions

- om_GetUnique()
  - Allocate a unique number to be used for a page; update the information in the page’s header; and return the unique number allocated.
  - Parameters
    - PageID *pid

      (IN) ID of the page for which a unique number is to be allocated
    - Unique *unique

      (OUT) The unique number allocated
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(…)
{
    PageID pid;     /* ID of a page */
    SlottedPage *apage;     /* pointer to a buffer */
    …
    /* Allocate a unique number to be used for the page */
    e = om_GetUnique(&pid, &(apage->slot[-i].unique));
    if (e < 0) ERRB1(e, &pid, PAGE_BUF);
    …
}
```

- om_FileMapAddPage()
  - Insert a page into the list of pages of a file.
  - Parameters
    - ObjectID *catObjForFile

      (IN) ID of the object storing the information about the file
    - PageID *prevPID

      (IN) ID of the page to become the previous page of the page to be inserted
    - PageID *newPID

      (IN) ID of the page to be inserted
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(
    ObjectID    *catObjForFile, /* IN: ID of the object that contains the catalog
information */
    ObjectID    *nearObj,   /* IN: create the new object near this object */
    ...)
{
    PageID pid;     /* ID of a page */
    ...
    /* Insert the page into the list of pages of the file */
    e = om_FileMapAddPage(catObjForFile, (PageID *)nearObj, &pid);
    if (e < 0) ERRB1(e, &pid, PAGE_BUF);
    ...
}
```

- om_FileMapDeletePage()
  - Delete a page from the list of pages of the file.
  - Parameters
    - ObjectID   *catObjForFile

      (IN) ID of the object storing the information about the file
    - PageID   *newPID

      (IN) ID of the page to be deleted
  - Return value
    - Four error code

– Example

```
Four EduOM_DestroyObject(
    ObjectID    *catObjForFile, /* IN: ID of the object that contains the catalog
information */
    ...)
{
     PageID pid;    /* ID of a page */
    ...
    /* Delete the page from the list of pages of the file */
    e = om_FileMapDeletePage(catObjForFile, &pid);
    if (e < 0) ERRB1(e, &pid, PAGE_BUF);
    ...
}
```

- om_PutInAvailSpaceList()
  - Insert a page into an available space list.
  - Parameters
    - ObjectID *catObjForFile

      (IN) ID of the object storing the information about the file
    - PageID *pid

      (IN) ID of the page to be inserted
    - SlottedPage *apage

      (INOUT) Page to be inserted
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(
    ObjectID    *catObjForFile, /* IN: ID of the object that contains the catalog
information */
    ...)
{
    PageID pid;     /* ID of a page */
    SlottedPage *apage;      /* pointer to a buffer */
    ...
    /* Insert the page into the available space list */
    e = om_PutInAvailSpaceList(catObjForFile, &pid, apage);
    if (e < 0) ERRB1(e, &pid, PAGE_BUF);
    ...
}
```

- om_RemoveFromAvailSpaceList()
  - Delete a page from an available space list
  - Parameters
    - ObjectID *catObjForFile

      (IN) ID of the object storing the information about the file
    - PageID *pid

      (IN) ID of the page to be deleted
    - SlottedPage *apage

      (INOUT) Page to be deleted
  - Return value
    - Four error code

– Example

```
Four eduom_CreateObject(
    ObjectID    *catObjForFile, /* IN: ID of the object that contains the catalog
information */
    ...)
{
    PageID pid;    /* ID of a page */
    SlottedPage *apage;     /* pointer to a buffer */
    ...
    /* Delete the page from the available space list */
    e = om_RemoveFromAvailSpaceList(catObjForFile, &pid, apage);
    if (e < 0) ERRB1(e, &pid, PAGE_BUF);
    ...
}
```

Error Handling

- Error handling macro
  - ERR(e)
    - Write the error code *e* given as a parameter, the file name, and the position where the error occurred into the error log file (odysseus_error.log); return the error code.
    - Usage example

      ```
      if(length < 0) ERR(eBADLENGTH_OM)
      ```
  - ERRB1(e, pid, t)
    - The same as ERR(e) except that it unfixes the page having the page ID, *pid*, given as the parameter before returning the error code *e*.
    - Usage example

      ```
      if(e < 0) ERRB1(e, &pid, PAGE_BUF)
      ```

- Error code

  See the $(EduOM_HOME_DIR)/Header/EduOM_errorcodes.h File.

How to Do the Project

- Files used in the project
  - Files that students are to implement
    - Skeleton file (.c file)

      Files containing the functions whose implementation part is omitted
  - Files given to students
    - Object file (.o file)

      File that ODYSSEUS/COSMOS, which is the underlying system, is compiled as an object file. It contains all the functions in ODYSSEUS/COSMOS storage system including the given lower-level functions that are called in the modules to be implemented.
    - Header file (.h file)

      Files containing the definition of data structures and function prototypes used in the modules to be implemented and the test module
    - Source code file of the test module

      Source code file of the test module to test the functions in the implemented module
    - Executable solution file

      Executable file showing a correct test result

- How to perform the project
  - Implement the functions in the skeleton files.
    - For the implementation, a variety of macros are available in the header files in the $(EduOM_HOME_DIR)/Header directory.
  - Use the command make clean (deleting the object files and DB volumes) and make (compiling) to compile the skeleton files implemented and link them with the given object file.
    - As a result of compiling and linking, an executable file is created to test functions of the modules implemented.
  - Compare the execution results of your executable file with those of the given executable solution file.

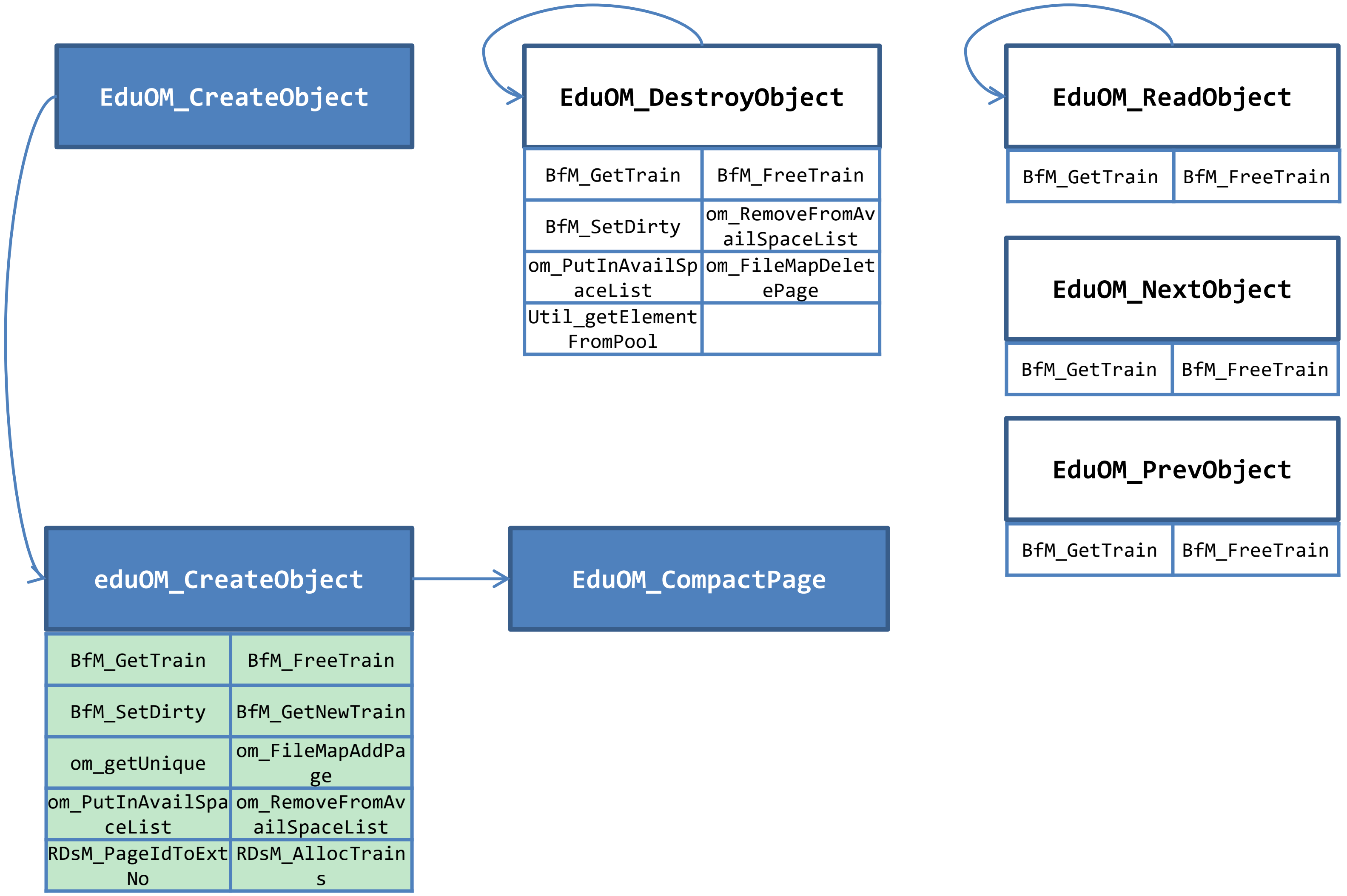
❖ How to test a function without implementing other functions
  - ❖ In the file $(EduOM_HOME_DIR)/Header/EduOM_TestModule.h,
    - ❖ For the API function that you have implemented, define the value of the corresponding macro to TRUE.
    - ❖ For the API function that you have not implemented, define the value of the corresponding macro to FALSE.
  - ❖ Enter the Make command to recompile the project.

❖ How to implement an API function without implementing some of its internal functions
  - ❖ Use the default solution function (internal function name with the prefix “edu” omitted).
    - ❖ e.g., the default solution function of the internal function eduom_CreateObject() is om_CreateObject().

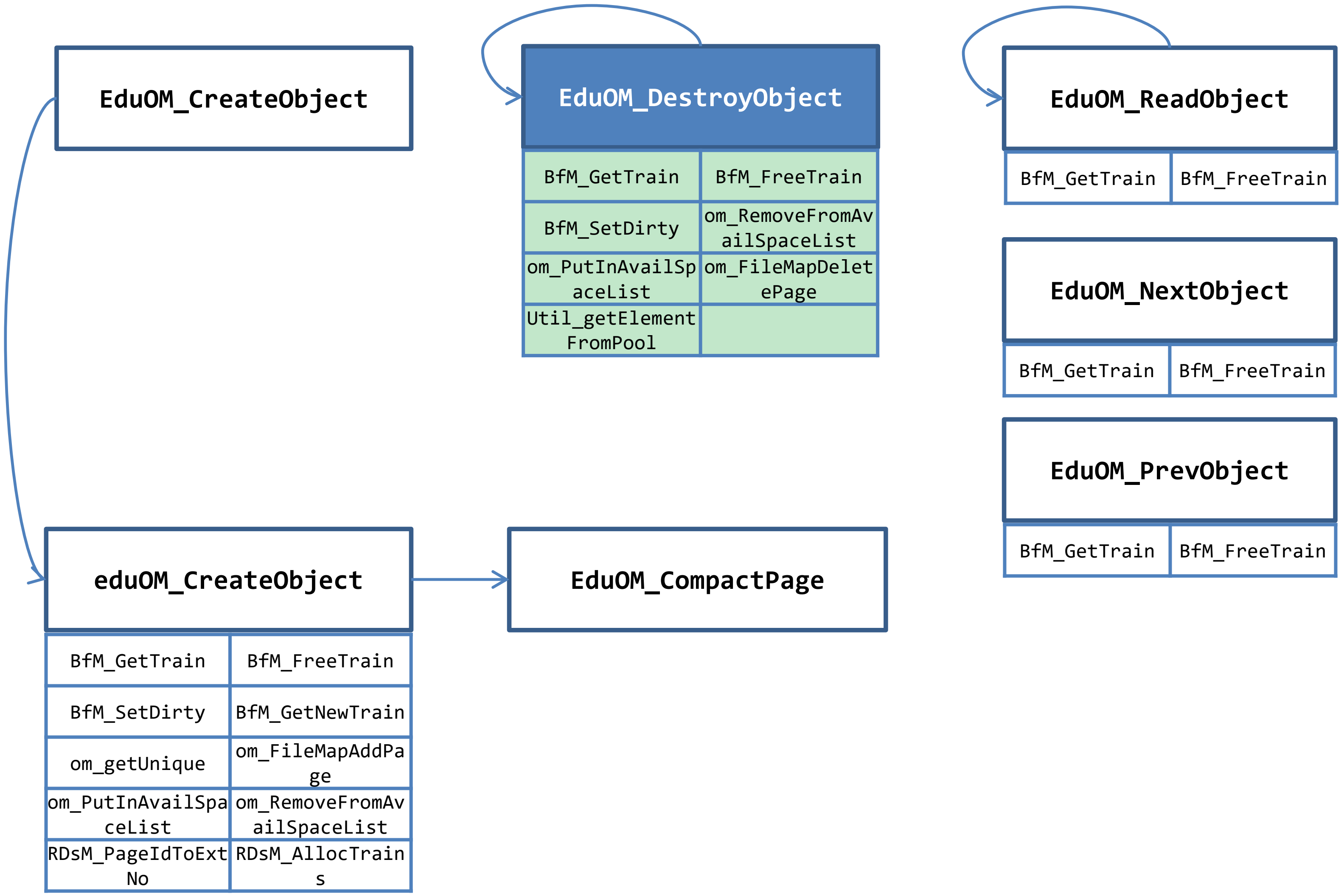
Appendix

Function Call Graph

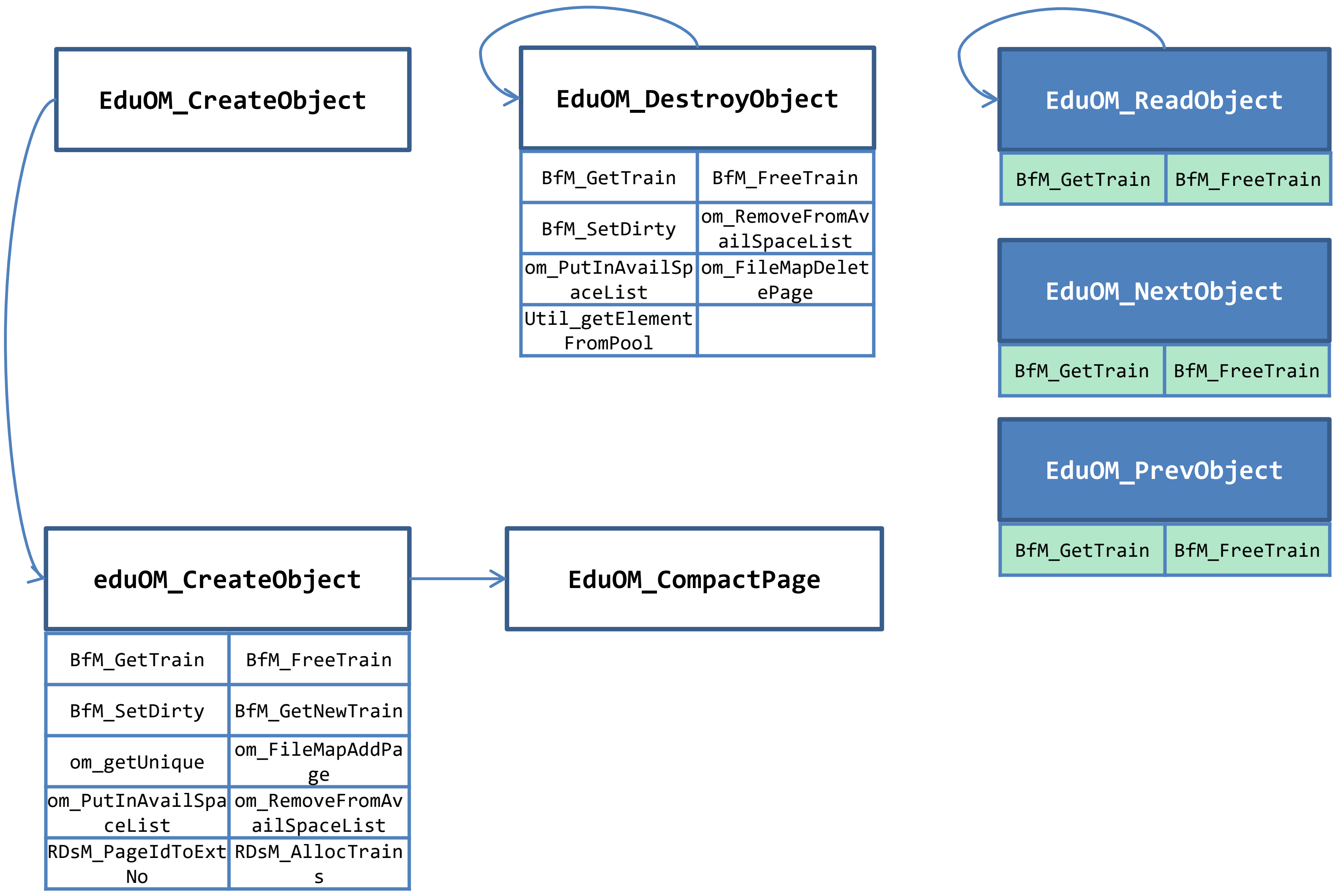
EduOM_CreateObject



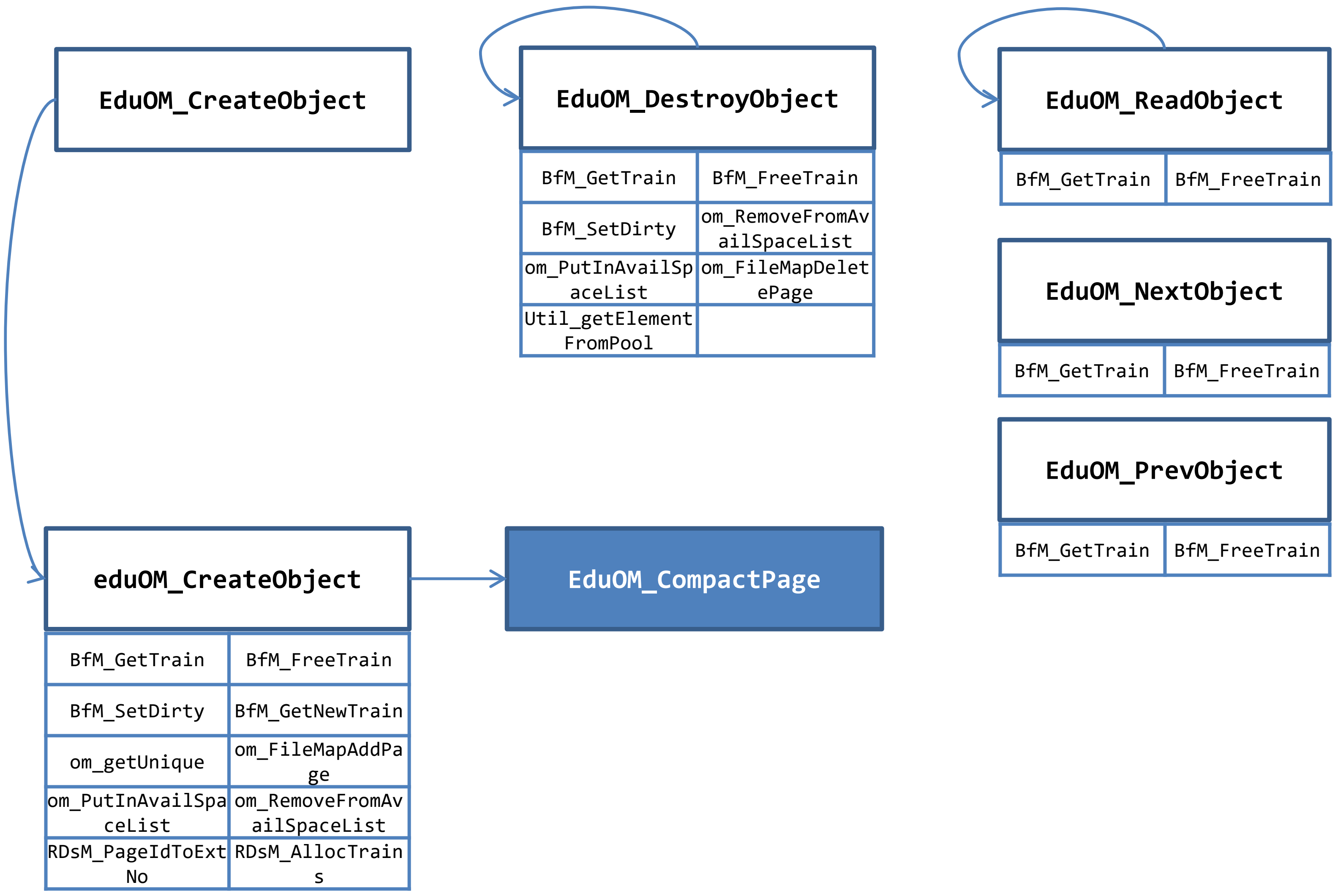
EduOM_DestroyObject



EduOM_ReadObject, EduOM_NextObject, EduOM_PrevObject



EduOM_CompactPage


EduOM_CreateObject
EduOM_DestroyObject
BfM_GetTrain
BfM_FreeTrain
BfM_SetDirty
om_RemoveFromAvailSpaceList
om_PutInAvailSpaceList
om_FileMapDeletePage
Util_getElementFromPool
EduOM_ReadObject
BfM_GetTrain
BfM_FreeTrain
EduOM_NextObject
BfM_GetTrain
BfM_FreeTrain
EduOM_PrevObject
BfM_GetTrain
BfM_FreeTrain
eduOM_CreateObject
BfM_GetTrain
BfM_FreeTrain
BfM_SetDirty
BfM_GetNewTrain
om_getUnique
om_FileMapAddPage
om_PutInAvailSpaceList
om_RemoveFromAvailSpaceList
RDsM_PageIdToExtNo
RDsM_AllocTrains
EduOM_CompactPage